The Design of the *OpenSTARS* Adaptive Analyzer for Real-Time Distributed Systems

Kevin Bryan, Tiegeng Ren, Jiangyin Zhang, Lisa DiPippo, Victor Fay-Wolfe
Computer Science and Statistics, University of Rhode Island
{bryank,rentg,zhang,dipippo,wolfe}@cs.uri.edu

Abstract

This paper describes the design of the OpenSTARS real-time analysis tool. The paper focuses on criteria for a good analysis tool including correctness, performance/scalability, flexibility, and extensibility. Several leading real-time analysis tools are surveyed and several problems with the tools under these criteria are identified. The paper then presents the basic components and operation of OpenSTARS and how its design addresses these problems.

1. Introduction

Developing software for real-time applications faces two significant challenges. First, the applications often interact with the physical world in critical ways, and thus require robust, well-analyzed software that is verified before it is deployed. Second, the requirement to meet timing constraints adds a new dimension to the development effort – a dimension that adds significant complexity. The need for robust software, and the complexity associated with creating it, means that good real-time software development tools are essential. There are several real-time development/analysis tools (PERTS/RapidRMA [1,2], TimeWiz[3], VEST[4], Cheddar[5], are some). While these tools do real-time analysis fairly well, in general, they were not designed with performance, scalability, flexibility, and extensibility in mind.

This paper describes OpenSTARS (Open Schedulability Tool for Analysis of Real-time Systems) being developed at the University of Rhode Island. OpenSTARS is designed to be an efficient, scalable, flexible, and extensible open-source tool and framework that real-time systems researchers and practitioners can use for both offline and online analysis, and into which they can insert their own scheduling and quality of service (QoS) management algorithms.

Section 2 of this paper elaborates on the criteria for good real-time analysis tools. Section 3 evaluates the above-mentioned real-time analysis tools using those criteria. Section 4 presents OpenSTARS and framework design. Section 5 shows how OpenSTARS addresses the criteria and the deficiencies in existing tools. Section 6 summarizes and outlines the further development of the OpenSTARS tool.

2. Criteria For Good Real-Time Analysis Tools

We use four classes of criteria: *Correctness, Performance / Scalability, Extensibility,* and *Usability* both to evaluate existing real-time analysis tools and to provide the design principals for OpenSTARS.

Correctness Criteria. The are two primary tests for the correctness of a real-time analysis tool:

1) If the system is not schedulable, the tool must determine that it is not schedulable.
2) All parameters, such as response times, priorities, and missed deadlines that result from the tool's analysis must be accurate.

Note that the inverse of Correctness Test 1 is not necessarily required – if the system is schedulable the tool does not always need to determine this. That is, the analysis is, at times, allowed to be pessimistic.

Performance and Scalability Criteria. Performance and scalability issues fall into three categories:

1) Minimizing the resources used during analysis.
2) Allowing user input to trade off thoroughness of analysis for speed of analysis.
3) Use of optimized algorithms to perform analysis.

Due to the complexity of real-time scheduling, the analysis of non-trivial systems can easily exceed the time or computing resources available to do the analysis. For instance, classic time demand analysis [6] that checks the interactions of tasks at selected time instances can take excessive time and resources if there is a wide variation in the length of the periods of the tasks. To achieve good performance, the tool should yield results as quickly as possible while consuming as few resources as possible. It should also allow the user to choose between speed and thoroughness of the analysis, perhaps allowing less than optimal analysis to be performed more quickly. To this end, the tool should be able to give the user hints as to how long an analysis will take. For scalability the analysis should take special care in how it manages the analysis, using algorithms that are computationally optimized, and by breaking models with many tasks, resources, and time intervals into smaller pieces that can be analyzed efficiently.

Flexibility Criteria. Flexibility is related to how much control the user has over the tool. There are two primary areas of control that a tool could provide:

1) Control of the algorithms and parameters of analysis.
2) Control of the interface.

If more than one algorithm is available in the tool, then a knowledgeable user should be able to select one or more of them to run, possibly with guidance from the tool. The tool should make these decisions completely for a user that does not want to make them. Similarly, the tool should expose parameters that affect the tool's analysis so that a knowledgeable user could set them. Another facet of algorithm control is the ability to extract the scheduling algorithms and analysis used in the tool to be used in contexts other than as a stand-alone tool. For example, it might be useful to be able to use the analysis "engine" in other projects such as on-line schedulers.

For interface flexibility, the tool should have a well-defined input and output file format so that the user is not forced to use a particular Graphic User Interface (GUI) to model the system. Many users already have a preferred modeling tool for their systems, such as a UML based tool, or a general modeling tool such as the Generic Modeling Environment (GME) [7]. The real-time analysis tool's use of a well-defined input and output file format will facilitate the custom design of the tool's interface, or its use with an existing modeling tool, or tool chain.

Extensibility Criteria. The tool should be able to be extended to allow for the inclusion of new analysis, scheduling, and QoS management algorithms. Since researchers are regularly producing new algorithms in these areas, it is important that the tool's framework and design accommodate their inclusion into the tool. Also, since many real-time applications require custom analysis that is unique to the application, the ability to easily extend the tool for domain-specific analysis can be important. Three primary desirable features to support extensibility are:

1) A well-defined interface for inserting new algorithms without knowing the internals of the tool.
2) A well-documented software engineering design of the tool including use cases, class diagrams, interaction diagrams, a standard testing procedure with test cases that support component testing, integration testing, and regression testing.
3) Open source code for those who need to know the internals of the tool.

3. Evaluating Existing Tools

Existing tools that we have investigated are described below:

- *RapidRMA* – is a comprehensive tool originating from the PERTS project at University of Illinois, and now owned by Tri-Pacific Software [1,2].
- *VEST (Virginia Embedded Systems Toolkit)* – developed at University of Virginia, uses an interface developed in GME. VEST provides a modeling environment for building software and mapping it to hardware, and interpreters for running non-functional tests such as real-time analysis [4].
- *Cheddar* – developed at the University of Brest, is an open source framework written in Ada that implements many real-time scheduling algorithms [5].
- *TimeWiz* – developed by TimeSys Corporation, is a comprehensive tool for real-time modeling and analysis. It has been integrated with Rational Rose, now owned by IBM. Unfortunately, we were not able to obtain a copy of TimeWiz to evaluate. Therefore our conclusions are based solely on widely available documentation [3].

In evaluating these tools, we picked a handful of special cases to test. These include: (*a*) a large task set where 200 tasks with a combined 95% utilization are put on a single node and the task set is not schedulable; (*b*) a pair of tasks with widely varying periods of 2 seconds and 1 day (86,400 seconds); (*c*) a task set in which two tasks span two nodes. The task characteristics are listed in Table 1.

Correctness Evaluation. All of the tools correctly identified that the first test case was not schedulable. (this was not the only correctness that we ran, but we use it here as a representative case.)

Some tools had problems with accuracy. In test case *b*, it is easy to see that the second task (T2) is preempted every two seconds, for one second, it will take twice as long (or 57600 seconds) to complete. Cheddar correctly produces this result. RapidRMA gives a result of 57611, which while this may be acceptable, it is pessimistic.

For test case *c*, we found that RapidRMA again leads to an unnecessarily pessimistic result. This appears to be because RapidRMA has assumed that due to the first two subtasks being of equal period, either one could go in either order, and therefore RapidRMA assumes that $T_{1,2}$ and $T_{2,2}$ must be assumed to start at 60. This causes the system to be non-schedulable. The problem with this is one task will always go first, even though we do not know which. Whichever one does go first (e.g., $T_{1,1}$), its second subtask ($T_{1,2}$) will be able to execute concurrently with the other task's first subtask ($T_{2,1}$, in our example). If RapidRMA had chosen distinct priorities for each task it could have found this result.

All of tools we could run gave correct results on the rest of the tests given here. We have found other errors in some of the other tools, but we are still in the process of discussing them with those vendors.

Performance and Scalability Evaluation. The preliminary testing results show that for test case *a*, RapidRMA takes approximately 30 seconds and 542 MB memory to find the result. Increasing the number of tasks to 300, caused RapidRMA to use 2.1 GB of memory, and 2.5 minutes (part of which is due to swapping. The system would only commit 1.5GB of physical RAM to the process). This non-linear resource usage raises questions of RapidRMA's scalability.

Cheddar requires a code change to support more than 100 tasks or 30 processors, and doing so increases its static data size. Once this change is made, however, it completes in a reasonable amount of time, assuming the tasks are spread over a number of processors. If there are more than 100 tasks, it starts to take significantly longer. It will complete 100 tasks in 1 minute, but 200 tasks took Cheddar 11 minutes. Note that RapidRMA had completed the 200 tasks set (test case *a*) in 30 seconds, probably due to some form of optimization.

Flexibility Evaluation. Cheddar provides a very nice simulation engine, however it has one significant drawback: It only allows maximum 1500 time units in a simulation. When doing a simulation of tasks with phases, you are not guaranteed to get the worst-case response time unless you simulate at least as far as the Least Common Multiple (LCM) of the periods plus the maximum phase. Unless the system only contains a few small periods, or many of the periods are harmonic, it is likely that the LCM will be greater than 1500.

For algorithm parameter selection, RapidRMA allows the user to change the priority assignment between Rate Monotonic and Deadline Monotonic. For systems that contain tasks with deadlines shorter than their periods, there is no indication to the user that choosing Rate Monotonic is unnecessarily pessimistic. Both RapidRMA and Cheddar allow a user to analyze a system with dependencies as if there are no dependencies. This can lead to confusing, and wrong, results.

Both RapidRMA and Cheddar run more than one schedulability test at once, but without letting the user specify which one(s) are preferred. VEST allows the user to select which algorithm to use.

Of the tools that we evaluated, only Cheddar provides a framework API that can be invoked as part of an online schedulability test. All other tools only provide offline analysis. Unfortunately, the Cheddar API does not provide for retrieving all of the vital information needed for implementing the schedule. Therefore none of the tools have extractable components that are suitable for an online scheduler.

All of the tools in this evaluation require using a GUI. While GUI's are generally helpful in interpreting the results and for single-use modeling, they can be limiting for using the tool as part of a tool chain or for performing a suite of analyses. Cheddar partially addresses this concern by using a simple XML format for describing the tasks, but Cheddar has only a GUI output. RapidRMA has proprietary schema files that describe their input and "saved" format but they are undocumented and difficult to interpret. VEST interprets a GME model directly, so its input file format is the same as GME and it provides a only a GUI output. It is not clear from the documentation on TimeWiz whether the input format is documented, but it appears that the input and output is graphical only. Of these tools only RapidRMA provides output parameters that might help the run-time system, such as the priorities it used when doing the analysis.

Extensibility Evaluation. As for extensibility, RapidRMA, and TimeWiz are closed source, and so they are not designed for user's to extend or modify. Cheddar is an open-source project, and it provides a framework for people to extend it with new scheduling algorithms or use it in other environments. Cheddar allows adding simulation algorithms at run-time in a specially designed language. However, since it is written in Ada, which limits its ease of extensibility for many users.

4. OpenSTARS Design

This section describes our design of OpenSTARS. The tool has both an offline implementation and an online implementation. The online and offline designs

Table 1. Test cases

(a) large task set with 200 tasks

Tasks	Period	Deadline	Exec time	Node
T1	2547	2547	9	0
T2	298	298	2	0
T…	…	…	…	0
T200	132	132	1	0

(b) tasks with extreme task period

Tasks	Period	Deadline	Exec time	Node
T1	2	2	1	0
T2	86400	86400	28800	0

(c) 2 end-to-end tasks with 2 subtasks span on 2 nodes

Tasks	Period	Deadline	Exec time	Node
$T_{1,1}$	100	100	30	0
$T_{1,2}$	100	100	30	1
$T_{2,1}$	100	100	30	0
$T_{2,2}$	100	100	30	1

share a unified data structure that represents the real-time system and the real-time analysis algorithms. The offline tool is decomposed into the following three major phases: *Setup*, *Analysis*, and *Result interpretation*. For the online scenario we assume that certain parts of the design will appear in different pieces of a larger system, but that there will be at least one piece that does the actual analysis.

4.1. Goals

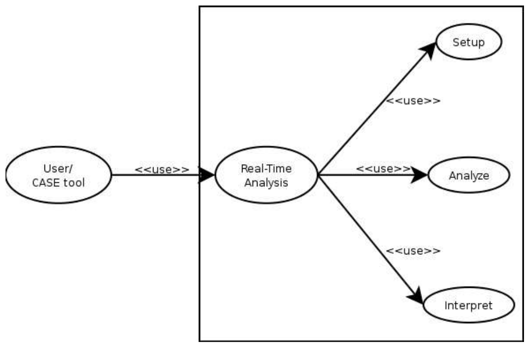
OpenSTARS has been designed to:

- Perform correct real-time analysis as defined in Section 2.
- Perform efficiently through optimizations in the analysis algorithms and optimizations in their implementation.
- Be scalable to allow for thousands of tasks and resources interacting over widely varying times.
- Be flexible by allowing knowledgeable users to compare scheduling and analysis algorithms in the tool.
- Be flexible by providing a tool "driver" that selects the proper analysis algorithms for users that want this level of support.
- Be flexible by using well-defined XML formats for both its input (see Appendix A) and output, thus facilitating OpenSTARS's integration with many GUIs and tool chains.
- Be flexible by supporting the extraction of modules for use in online run-time analysis and scheduling, thus providing consistency between offline analysis and run-time enforcement.
- Be extensible through the release of the source code and through use of procedures that allow researchers and practitioners to contribute new modules back to the tool repository.
- Be extensible by providing an API for those who do not wish to dig into source code to be able to insert new scheduling and analysis algorithms.
- Be extensible through use of a complete software engineering model and complete documentation as part of the open source release.
- Be easier to use by supporting domain-specific components that facilitate constructing models in the system. Example components are forms of networks, servers, distributable threads, data distribution, and message passing that are already modeled and kept in a library for inclusion into user models.

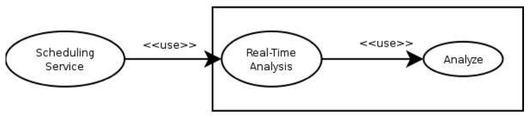
4.2 OpenSTARS Offline System Architecture

This section describes the basic components of OpenSTARS, then it describes the phases of its operation.

The system is designed for both online and offline use. The main difference between these two scenarios is that in the offline case, input and output are in XML file format, while in the online case, input and output are in form of message calls.



(a) Offline



(b)Online

Figure 1: Offline and Online Tool Design

4.2.1 Basic OpenSTARS Tool Components

The basic components of OpenSTARS are: the *Domain Object* which generalizes special purpose domain models to facilitate their inclusion; the *SchedAlgorithm Object*, which is the encapsulation of the scheduling algorithm that the application will use;

the *Driver Object*, which works with the user to selectively either expose choices or make choices for the user in the analysis process; and a *Result* object which contains the outcome of the analysis.

Domain Object. A Domain object is an implementation of an abstract base type for a particular sub-domain of a real-time system. This allows for a more natural representation of parts of the system. The interface for Domain objects includes a translation to and from the basic *Task*, *Resource*, and *Dependency* model (TRDModel) [8], which is the only format recognized by the scheduling algorithms.

Domain objects include the following important methods: *DomainToTRD*, *TRDToDomain*, *Interpret*, *Parse*, and *Key*. *Key* simply returns the name of the domain as it is expected to appear in the XML input file. The *Parse* method takes an XML structure and parses all of the data in it into its native representation. *DomainToTRD* converts this representation into a *TRDModel*, and *TRDToDomain* converts back again. *Interpret* is given the result of the analysis so that it can perform a mapping from the information contained therein to its own *Result* object.

SchedAlgorithm Object. The *SchedAlgorithm* object is also an abstract class that will be implemented for a particular real-time analysis algorithm. The interface for *SchedAlgorithm* objects includes a set of functions that take a list of characteristics of the system, and report: (a) whether the algorithm will return a correct result, (b) how quickly it can return a result, (c) what the likelihood of the algorithm returning an optimal result is, and (d) what information it will provide (in terms of priorities, worst case response times, utilizations, etc).

Initially, we are implementing a subset of the common algorithms. These include:

- *Utilization Function*: This calculates the utilization of a node to determine if a node is schedulable.
- *Optimized Time Demand Analysis*: This is generally the same as the Time Demand Analysis given in [6], but with several algorithmic optimizations that make difficult special cases easier [11].
- *Deadline Monotonic Analysis with Offset* [9]: This algorithm handles tasks with offsets more optimally than Time Demand, but at a significant cost in time.
- *Simulation*: This runs a Simulation of the tasks to determine worst-case response time. Depending on the periods of the tasks, this can also take a long time.

Driver Object. The *Driver* object is the interface between a real-time system model and a scheduling algorithm. The *Driver* acts as an intelligent agent by using adaptive strategies in selecting the best algorithm based on certain speed/optimality criteria. It is often the case that a given real-time system could be analyzed correctly by several algorithms, however each has their own time complexity and optimality percentage for it. Therefore, the Driver is used to rank the estimates on the amount of work by comparing the complexity analysis result of each of the algorithms.

Result Object. The *Result* object is an abstract class that holds the scheduling analysis result and domain information. Result in terms of sub-domain representation needs to be extracted from the TRDModel, since even the same result lead to different meaning in different sub-domain. For example, in an end-to-end scheduling analysis, a worst-case response time bigger than period is considered non-schedulability, however, in the Data Distribution System (DDS) the result is schedulable as long as worst-case response time is less than the deadline.

Moreover, from a usability point of view, users generally want more information on their real-time system than merely the schedulability. We designed the Result object as a holder for runtime information, such as system utilization, etc. Also, in case if a node is not schedulable, the Result object could also hold in-depth hints/suggestions like how we could change the task set to make it schedulable.

4.2.2 OpenSTARS Operation

We now describe the three phases of the offline OpenSTARS tool: *setup, analyze*, and *interpret*. Figure 2 gives a diagram that shows the major participants in each phase.

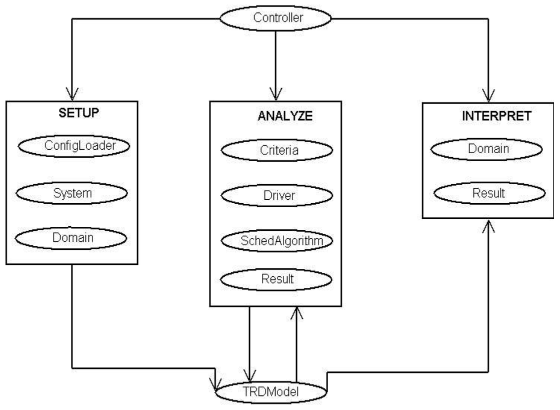


Figure 2. Top-level system architecture

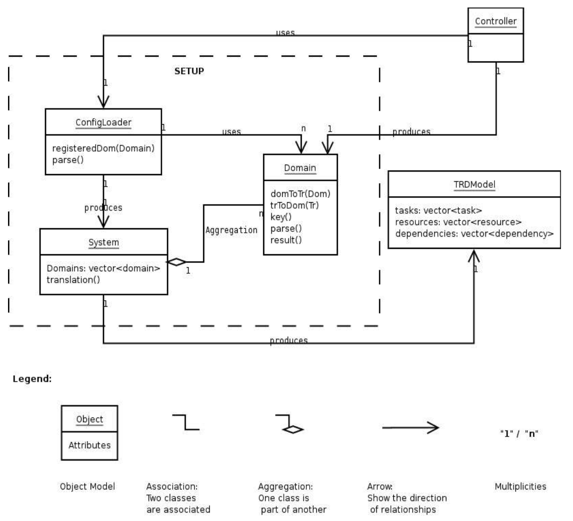


Figure 3. Object model for SETUP

Setup. During the first phase, the *Controller* creates *Domain* objects (one per Domain) and adds them to the System. *Domains* are registered with the *ConfigLoader* with a *Key* corresponding to the top level XML tag that it knows how to parse. As the configuration file loaded, each section is passed to the *Domain* objects. After all XML parsing is complete, the *Domains* then do a translation between their representation and the basic TRDModel.

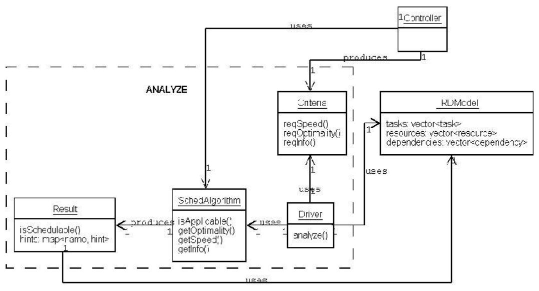


Figure 4. Object model for ANALYZE

Analyze. In the second phase, the *Driver* scans the TRDModel collecting statistics about the set of tasks and resources. The *Driver* then asks each algorithm four "questions" (the last three being dependent on the first):

1. *isApplicable* – Will the algorithm be able to give a correct result?
2. *getSpeed* – How quickly will the algorithm be able to work?
3. *getOptimality* – How close to optimal will the solution it gives be?
4. *getInfo* – What information will the algorithm give?

The *Driver* then builds a table with these answers and sorts and compares them according to the criteria given by the user (in this case, given by the command line) so that the *Driver* can choose the correct algorithm to run. The Driver then calls the *SchedAlgorithm* it chose, which will return a *Result* object. After that, the system passes that result to each *SchedAlgorithm* to allow it to map information back to its own structures.

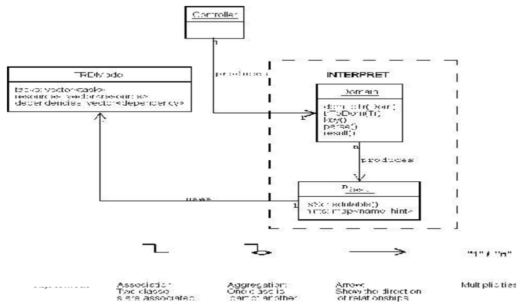


Figure 5. Object model for INTERPRET

Interpret. The *Controller* hands the TRDModel and the *Result* to each *Domain* object in the system. The *Domain* object maps the TRDModel back to its own data structures and adds more information to the *Result*. The *Controller* then asks each *Domain* object to save and output the result. Optionally, the system will ask each Domain object to output its result to an XML file; similarly it may ask the TRDModel to write itself to an XML file.

4.3 OpenSTARS Online System Architecture

For the online analysis the steps are almost identical with the following exceptions. Instead of reading information from an XML file, there will be an external service (for instance an online scheduling service), which maintains its own *Domain* model. The online OpenSTARS tool will then ask that *Domain* model to do a transformation to the TRDModel. (like the Initialization step in the offline tool process). In the *analyze* portion, the external service will provide the criteria. Output of online analysis will be in form of a data structure that is passed back to the caller with updated schedule information including: a yes/no answer to whether the system is schedulable, parameters and possible hints or other meta-data. In addition, the online analysis will generate a log of each transaction.

5. Evaluating OpenSTARS

In this section we evaluate OpenSTARS according to the criteria specified in Section 2. While the implementation of the tool is not complete, we evaluate it based on the design and current implementation.

OpenSTARS Correctness. OpenSTARS has a well-defined testing procedure. First is a suite of test cases that address special cases and have been verified by hand. The tests described in Section 3 are a few examples. Second, we have test cases that have been run through other tools to compare their results to those obtained by OpenSTARS. Third, due to OpenSTARS support for multiple analysis and scheduling algorithms, we can test one algorithm in OpenSTARS by using other algorithms in OpenSTARS. For example, by doing a simulation, it is possible to determine the exact worst-case response time, which can be compared against algorithmically determined analysis.

We have run the current OpenSTARS implementation against all of the tests described in Table 1 and it provides correct results for any test sets without dependencies. We are currently adding support for task dependencies.

OpenSTARS Performance and Scalability. OpenSTARS addresses performance on two fronts. First, it gives the user the ability to specify how long they are willing to wait for a response, and how thorough that they want the response to be. This is used by the OpenSTARS Driver to choose the correct scheduling algorithm. Secondly, OpenSTARS employs optimizations that we have developed that are not incorporated in other tools. For example, whereas most tools require about 16 calculations to compute the example that we gave earlier with a task of a period of one day, OpenSTARS does it in 2 calculations. OpenSTARS can analyze task sets of 200 tasks in 3 megabytes of memory and 1 second due to the optimizations that we've applied to the classic Time Demand Analysis [11]. This is much faster and less memory usage than other tools, as we described in Section 3.

OpenSTARS Flexibility. For flexibility, OpenSTARS gives the user the ability to select parameters such as speed and thoroughness, as well as providing information that helps select which algorithm it uses to determine schedulability. OpenSTARS also supports use of the offline tool components in an online environment through modularization of those components. OpenSTARS facilitates use with many GUIs and tool chains through a very simple yet comprehensive XML input and output format that is shown in Appendix A.

OpenSTARS Extensibility. There are several ways that OpenSTARS supports extensibility. The first is in facilitating domain specific representations through the use of the Domain object described in Section 4.2. For algorithm support, OpenSTARS uses a generic interface for all of the schedulability analysis algorithms. In a future revision, OpenSTARS may package algorithms as libraries instead of monolithically, to allow better control over the algorithms that are available at runtime.

The OpenSTARS project is open source, so others can extend it. To facilitate OpenSTARS's extension by other researchers and practitioners, we also have compiled documentation at each stage of the design. Furthermore, the OpenSTARS test suite is available in a standard form to facilitate, unit, regression and performance testing.

6. Conclusion

This paper has described the design of the OpenSTARS real-time analysis tool being developed at the University of Rhode Island. The tool is meant to be the basis of open-source project to which real-time researchers around the world can contribute. Our efforts have focused on a sound software engineering model for the tool, clear source code, a well-defined external interface and internal interfaces among tool components, and efficient, optimized base algorithms. This emphasis is yielding a tool that is not only correct, scalable, and flexible, but is also easily extensible by other researchers and practitioners in the true spirit and practice of open source projects.

We hope that by having a common implementation of each of the scheduling algorithms, that people will be encouraged to examine them and look for ways to optimize the calculations done. We have found several optimizations to Time Demand Analysis, that allow it to handle large variance in periods more gracefully, as well as some simplifications that can be made for use in an online environment. There is a minor change to the algorithm in [9] to reduce the amount of work done, however we have not yet evaluated how much impact this change will have.

We are currently setting up the OpenSTARS open source repository. This repository is modeled after the university-hosted ACE portable framework and TAO middleware open source repository [10]. The OpenSTARS repository will allow obtaining source code, test cases, complete design documentation, and for others to contribute new components to OpenSTARS.

Appendix A

```
<?xml version="1.0" encoding="UTF-8"
      standalone="no" ?>
<OpenSTARS>
  <!-- Specify a machine as a node -->
  <node name="node0">
    <!-- Specify resources available on
       this machine, such as cpu and mem-
       ory -->
    <resource active="yes" name="cpu1"
       preemptable="yes" rate="11"
       type="cpu"/>
    <resource active="yes" name="cpu2"
       preemptable="yes" rate="2"
       type="cpu"/>
    <resource active="no" name="mem1"
       preemptable="no" rate="1"
       type="memory"/>
  </node>
  <node name="node1">
    <resource active="yes" name="cpu1"
       preemptable="yes" rate="11"
       type="cpu"/>
    <resource active="no" name="mem1"
       preemptable="no" rate="1"
       type="memory"/>
    <resource active="no" name="disk1"
       preemptable="no" rate="1"
       type="disk"/>
  </node>
  <!-- Specify a task graph.  Note all
       subtasks will share the same peri-
       od -->
  <end2endtask deadline="40"
       importance="1" name="e2e1"
       period="50" phase="5">
    <!-- Specify a subtask. Node/active
       pair gives processor for this sub-
       task -->
    <subtask active="cpu1" deadline="0"
       exection_time="10" name="sub11"
       node="node0">
      <successors>sub12</successors>
    </subtask>
    <subtask active="cpu1" deadline="0"
       exection_time="20" name="sub12"
       node="node1">
       <!-- Specify dependencies.  Can be
       specified backwards or forwards;
       consistency check done at runtime.
       -->
      <successors>sub13</successors>
      <predecessors>sub11</predecessors>
      <!-- Specify resource acquisition
       and release. -->
      <resource_usage acq_time="3"
       deacq_time="5" name="mem1"/>
    </subtask>
    <subtask active="cpu1" deadline="0"
       exection_time="10" name="sub13"
       node="node1">
      <predecessors>sub12</predecessors>
      <resource_usage acq_time="1"
       deacq_time="2" name="disk1"/>
      <resource_usage acq_time="1"
       deacq_time="2" name="mem1"/>
    </subtask>
  </end2endtask>
  <!-- Specify a network topology -->
  <network>
    <!-- Specify a switch/hub/router -->
    <lan bandwidth="100"
       name="lan1">node0,node1</lan>
    <!-- Specify a direct link -->
    <link bandwidth="100" pointa="node0"
       pointb="node1"/>
  </network>
</OpenSTARS>
```

References.

[1] J. W. S. Liu, J. Redondo, Z. Deng, T. Tia, R. Bettati, A. Silberman, M. Storch, R. Ha, and W. Shih, "PERTS: A prototyping environment for real-time systems, Technical Report: UIUCDCS-R-93-1802, 1993.

[2] Tri-Pacific Software Inc. – Rapid RMA Data Sheet http://www.tripac.com/html/prod-fact-rrm.html

[3] Time Sys Corporation. – TimeWiz Data Sheet http://www.timesys.com/files/prodlit/ TimeWiz%20Data%20Sheet.pdf

[4] J. A. Stankovic, R. Zhu, R. Poornalingam, C. Lu, Z. Yu, M. Humphrey, and B. Ellis, "VEST: An Aspect-Based Composition Tool for Real-Time Systems," *The 9th IEEE Real-Time and Embedded Technology and Applications Symposium*, Toronto, Canada, May 2003.

[5] F. Singhoff, J. Legrand, L. Nana, and L. Marcé, "Cheddar: a Flexible Real Time Scheduling Framework," *ACM SIGADA Conference*, Atlanta, November 2004. http://beru.univ-brest.fr/~singhoff/cheddar/

[6] J. P. Lehoczky, L. Sha, and Y. Ding, "The rate-monotonic scheduling algorithm: Exact characterization and average case behavior," *Proceedings of Real-Time Systems Symposium*, pp. 166-171, December 1989.

[7] A. Ledeczi, M. Maroti, A. Bakay, G. Karsai, J. Garrett, C. Thomason IV, G. Nordstrom, J. Sprinkle, and P. Volgyes, "The Generic Modeling Environment," *Workshop on Intelligent Signal Processing*, Budapest, Hungary,2001.

[8] J. W. S. Liu, *Real-time Systems* ISBN 0-13-099651-3, pp.134, 2000.

[9] N. C. Audsley, *Optimal Priority Assignment and Feasibility of Static Priority Tasks With Arbitrary Start Times*, tech. report YCS164, Dept. Computer Science, University of York (1991).

[10] Institute for Software Integrated Systems, Vanderbilt University The ACE ORB. April 2004, http://escher.isis.vanderbilt.edu/tools/get_tool?TAO

[11] K. Bryan, and T. Ren, *"Optimizations on Time Demand Analysis for Real-Time Schedulability Test,"* tech. Report TR05-302, Dept. Computer Science, University of Rhode Island, 2005.